JOURNAL OF COMPUTER AND SYSTEM SCIENCES **52**, 170–184 (1996)
ARTICLE NO. 0013



ANDRZEJ EHRENFEUCHT,* JOOST ENGELFRIET,[†, 1] AND GRZEGORZ ROZENBERG*, [†, 1]

*Department of Computer Science, University of Colorado at Boulder, Boulder, Colorado 80309;
[†]Department of Computer Science, Leiden University, P.O. Box 9512, 2300 RA Leiden, The Netherlands

Received April 22, 1994; revised August 24, 1995

We introduce a new way of specifying graphs: through languages, i.e., sets of strings. The strings of a given (finite, prefix-free) language represent the vertices of the graph; whether or not there is an edge between the vertices represented by two strings is determined by the pair of symbols at the first position in these strings where they differ. With this new, ''positional'' or lexicographic, method, classical and well-understood ways of specifying languages can now be used to specify graphs, in a compact way; thus, (small) finite automata can be used to specify (large) graphs. Since (prefix-free) languages can be viewed as trees, our method generalizes the hierarchical specification of particular types of graphs such as cographs and VSP graphs. Our main results demonstrate an intrinsic relationship between the fundamental operations of language concatenation and graph substitution. 

## INTRODUCTION

Many classes of graphs can be defined in an inductive way, i.e., as the smallest class of graphs containing certain elementary graphs and closed under certain graph operations. This means that every graph in the class can be represented by an expression, or, equivalently, by a labeled tree. Well-known examples are the class of cographs (represented by cotrees) [CorLerSte], the class of minimal (or, transitive) vertex series–parallel graphs (represented by binary decomposition trees) [ValTarLaw], and the class of graphs of tree-width $\leqslant k$ [RobSey]. The advantage of representing graphs by trees is that properties of graphs can be verified by induction on the tree, often leading to efficient algorithms (see, e.g., [CorLerSte, ValTarlaw, AdhPen, BodMöh, Arn, ArnLagSee, Cou3, EngHarProRoz]).

The idea of representing cographs by cotrees (or transitive VSP graphs by binary decomposition trees) has been generalized to arbitrary graphs: every graph can be represented by a tree that expresses its "clan structure," where a clan (or module, or clumping, or autonomous set, or...; see, e.g., [BueMöh, MöhRad]) is a set of vertices of the graph such that any vertex outside the set is either connected to all vertices in the set or to none. In [MulSpi] the tree is called the modular decomposition of the graph, in [MöhRad] it is called its composition tree, and in [EhrRoz2] it is called its shape or prime tree family (and the concept is defined for so-called 2-structures that are more general than graphs).

Rather than labeling each vertex of a tree (by a graph operation) one can, equivalently, label the outgoing edges of the vertex (by the graph operation and an argument selector). Edge labeled trees, such that all edges leaving a vertex of the tree have distinct labels, are in one-to-one correspondence with (finite) prefix-free languages over the alphabet of edge labels. In fact, the language corresponding to a tree consists of all label sequences of paths from the root to the leaves of the tree. The basic idea in this paper is to turn the representation of graphs by trees into the representation of graphs by languages, using the above correspondence between trees and languages. Thus, we investigate the representation of graphs by prefix-free languages (where both the graphs and the languages are finite). Since, in cotrees, binary decomposition trees, modular decompositions, composition trees, and shapes, the leaves of the tree represent the vertices of the graph, we now let the strings of a given prefix-free language represent the vertices of the graph. To define the edges between such vertices we assume the alphabet of the language to have a graph structure, and we connect two vertices, represented by two strings, if the first two symbols at which the strings differ are connected in the alphabet graph. This positional or lexicographic idea of "first difference" is in accordance with cotrees, binary decomposition trees, and shapes (because two paths from the root to two leaves part at the least common ancestor of those leaves). It is considered implicitly at the end of [Sab2].

Representing graphs by languages yields the possibility to use ideas and concepts from formal language theory to investigate graphs. In particular we investigate how the structure of the language influences the structure of the graph. One way of structuring a language is by building it as a concatenation of other, simpler languages. The operation of concatenation is certainly the most basic and well-understood operation in formal language theory. The main

[1] The research of these authors has been supported by the Esprit Basic Working Groups COMPUGRAPH (No. 3299) and ASMICS II (No. 6317).

0022-0000/96 $12.00

outcome of our investigations is the close connection between concatenation of languages and substitution of graphs. The operation of graph substitution that we consider, consists of the substitution of graphs for the vertices of a given graph, in such a way that the substituted graphs become clans of the resulting graph. This natural operation of substitution of graphs is well known in graph theory (see, e.g., [Har, Gol], where it is called graph composition). Its close correspondence to the representation of graphs by (de)composition trees is stressed in [MöhRad]. Generalizations of it are extensively used in the area of graph grammars [EhrKreRoz]. The relationship between concatenation of languages and substitution of graphs will be expressed in several ways. In Section 3 we show that every graph that is represented by a language over a given alphabet, can be obtained by repeated substitution into the subgraphs of the alphabet (recall that it is assumed that the alphabet is a graph). In Section 4 the concatenation of two languages is shown to correspond to the substitution of one graph for each vertex of another graph.

Section 1 contains preliminaries, including the definition of graph substitution. In Section 2 we present the main definition of how a language represents a graph, and we establish some basic properties of this representation. In particular, it is shown that quotients of the language (in the formal language theoretic sense) give clans in the graph. The operation of taking the quotient (or derivative) of a language (by a string) is a classical language theoretic operation: according to the well-known Nerode theorem the set of quotients of a language defines the minimal deterministic finite automaton that recognizes the language (see, e.g., [Eil, Section III.5] or [RabSco, Brz]). Moreover, in terms of trees this minimal automaton is the graph obtained from the tree, corresponding to the language, by sharing equal subtrees (due to the finiteness and prefix-freeness of the language). This leads to the idea of representing graphs by minimal finite automata or, equivalently, by trees with shared subtrees, a representation that is more compact in general than by trees. It should be clear that concatenation of, say, two languages is useful for this compactness; the two minimal automata can just be connected in sequence, whereas in the tree representation many copies would have to be made of the tree of the second language. It should also be clear (although it will not be investigated in this paper) that efficient graph algorithms that work on the tree representing the graph can as well work on the tree with shared subtrees.

We assume the reader to be familiar with elementary concepts from formal language theory (see, e.g., [HopUll]).

## 1. GRAPHS, CLANS, AND SUBSTITUTION

We consider ordinary loop-free directed graphs $g=(V, E)$, where $V$ is the finite nonempty set of vertices and $E \subseteq E_2(V)$ is the set of edges, with $E_2(V) = (V \times V) - \{(x, x) \mid x \in V\}$. However, except in examples, we will view $E$ as a boolean function $E_2(V) \to \{0, 1\}$, with $E(x, y) = 1$ iff $(x, y) \in E$ for all distinct $x, y \in V$. We do this for two reasons: (1) it is technically more convenient, and (2) all our results can easily be generalized to "labeled 2-structures," which are pairs $(V, E)$ where $E$ is a mapping $E_2(V) \to \Delta$, and $\Delta$ is an arbitrary finite set (see [EhrRoz1, EhrRoz2] for 2-structures, and in particular [EhrRoz2, Section 6] for labeled 2-structures).

For a graph $g$, we denote its components by $V_g$ and $E_g$.

As usual, for two graphs $g$ and $g'$, an *isomorphism* from $g$ to $g'$ is a bijection $\phi: V_g \to V_{g'}$, such that $E_{g'}(\phi(x), \phi(y)) = E_g(x, y)$ for all distinct $x, y \in V_g$; $g$ and $g'$ are *isomorphic*, denoted $g$ **isom** $g'$, if there is an isomorphism from $g$ to $g'$. We would like to point out that in this paper we do *not* identify isomorphic graphs formally, as is often done, for example in the area of graph grammars. Next, as usual, $g'$ is an induced subgraph of $g$ if $V_{g'} \subseteq V_g$ and $E_{g'}(x, y) = E_g(x, y)$ for all distinct $x, y \in V_{g'}$; since we will consider induced subgraphs only, we will say shortly that $g'$ is a *subgraph* of $g$. For $X \subseteq V_g$, the subgraph of $g$ induced by $X$ will be denoted $g[X]$, or just by $X$ if it is clear from the context that the subgraph is meant rather than the set.

We will investigate compact representations of graphs that have a very regular structure. In general, this regularity will be caused by the presence of many isomorphic clans in the graph. Let $g=(V, E)$ be a graph, and let $X$ be a nonempty subset of $V$. $X$ is a *clan* of $g$ if, for every $x, y \in X$ and $z \in V - X$, $E(x, z) = E(y, z)$ and $E(z, x) = E(z, y)$. If $X$ is a clan, we will also say that the subgraph $g[X]$ induced by $X$ is a clan. Well-known (and easily provable) facts about clans are the following (see, e.g., [EhrRoz1, MöhRad]). The set $V$, and all singleton sets $\{x\}$, $x \in V$, are clans: the *trivial* clans. The intersection of two (nondisjoint) clans is a clan. If $X$ and $Y$ are disjoint clans, then, for every $x, x' \in X$ and $y, y' \in Y$, $E(x, y) = E(x', y')$.

A natural way to construct a graph with many clans is by substituting graphs for the vertices of a graph, as follows. Let $g$ be a graph, and let $g_x$ be a graph for every $x \in V_g$. Intuitively, we substitute $g_x$ for vertex $x$ in $g$, in such a way that the edges between $g_x$ and the rest of the resulting graph are inherited from the edges between $x$ and the rest of $g$; in this way $g_x$ becomes a clan of the resulting graph. Formally, the *substitution* of $g_x$ for $x$ into $g$, denoted by $g[x \leftarrow g_x]_{x \in V_g}$ or just $g[x \leftarrow g_x]$, is the graph $(V, E)$ with

$$V = \{(x, y) \mid x \in V_g, y \in V_{g_x}\},$$

and for distinct $(x_1, y_1), (x_2, y_2) \in V$,

$$E((x_1, y_1), (x_2, y_2)) = \begin{cases} E_g(x_1, x_2) & \text{if } x_1 \neq x_2 \\ E_{g_x}(y_1, y_2) & \text{if } x_1 = x_2 = x. \end{cases}$$

Defining the vertices of the resulting graph as ordered pairs of vertices of the component graphs (as in [Har, Sab1, Sab2, Sab3]) turns out to be technically very convenient for the purposes of this paper (see, in particular, the statements of Theorem 4 and Theorem 12).

It should be clear from the above definition that, for every $x \in V_g$, $\{(x, y) \mid y \in V_{g_x}\}$ is a clan of $g[x \leftarrow g_x]_{x \in V_g}$ that is isomorphic (as a subgraph) with $g_x$. Thus, these sets form a partition of $V$ into clans. It should also be clear that substitution behaves correctly with respect to isomorphism, i.e., if $g'_x$ **isom** $g_x$ for all $x \in V_g$, then $g[x \leftarrow g'_x]$ **isom** $g[x \leftarrow g_x]$. Also, if $\phi$ is an isomorphism from $g$ to $g'$, then $g'[\phi(x) \leftarrow g_x]$ **isom** $g[x \leftarrow g_x]$.

As an example of substitution, consider the graph $g = (V, E)$ of Fig. 1(a) with $V = \{u, v, y, z\}$ and $E = \{(u, v), (v, y), (y, z), (u, z)\}$. For the vertices $x$ of $g$ we substitute the graphs $g_x$ shown in Fig. 1(b), i.e., for $v$ the graph with two vertices and one edge, for $y$ the discrete graph with two vertices, for $u$ the one-vertex graph, and for $z$ the graph with vertices $z_1, z_2, z_3$ and edges $(z_1, z_2)$ and $(z_3, z_2)$. The result of the substitution, $g[x \leftarrow g_x]_{x \in V}$, is shown in Fig. 1(c), where we have indicated vertex $(v, v_1)$ simply by $v_1$ and, similarly, for the other vertices.

Note that, intuitively, in a substitution $g[x \leftarrow g_x]$ one does not have to substitute a graph for *every* vertex of $g$, in the sense that one can always take $g_x$ to be the one-vertex graph, the substitution of which does not change the graph (as for $u$ in the previous example). The special case of $g[x \leftarrow g_x]$, where all but one $g_x$ are the one-vertex graph was considered in [EhrRoz2, Definition 7.9], where the connection of this type of substitution to the substitution operation in certain graph grammars was pointed out.

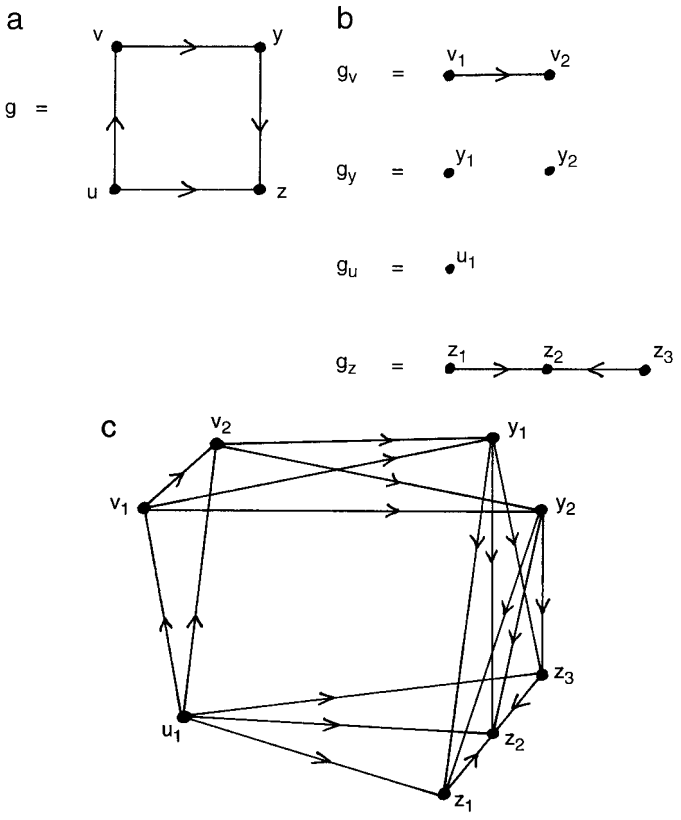


**FIG. 1.** Graph Substitution.

## 2. LANGUAGES THAT REPRESENT GRAPHS

A language $L$ is a set of strings, where each string is a sequence of symbols from some alphabet. In order to let $L$ represent a graph $g$, we will assume that the alphabet also has a graph structure, with the symbols as vertices. The basic idea is to let the strings of $L$ be the vertices of $g$ and to put an edge between two strings in $g$ iff, in the alphabet graph, there is an edge between the two symbols at the position in the strings where they first differ from each other. In order that this "first difference" always exists, we require the language $L$ to be prefix-free. Since a prefix-free language can be viewed as a tree, our approach is in line with well-known ways of representing graphs by trees, as discussed in the Introduction.

Let $V$ be an alphabet, and let $V^*$ denote the set of all strings over $V$, as usual. The empty string is denoted $\lambda$. For strings $x, y \in V^*$, $x$ is a *prefix* of $y$ if $y = xz$ for some $z \in V^*$. A language $L \subseteq V^*$ is *prefix-free* if there are no distinct $x, y \in L$ such that $x$ is a prefix of $y$. For a language $L \subseteq V^*$ and a string $x \in V^*$, $x$ is a *prefix* of $L$ if $x$ is a prefix of $y$ for some $y \in L$.

DEFINITION 1. Let $h$ be a graph. A *graph representation language*, abbreviated *grep language*, *over* $h$ is a nonempty finite prefix-free subset $L$ of $V_h^*$. The *graph defined by* $L$, denoted $\mathrm{gra}_h(L)$ or just $\mathrm{gra}(L)$ if $h$ is clear from the context, is the graph $(V, E)$ with $V = L$ and for distinct $x, y \in L$, $E(x, y) = E_h(a, b)$, where $a, b \in V_h$ are the first symbols of $x$ and $y$, respectively, where $x$ and $y$ differ (i.e., $x = uax'$ and $y = uby'$ for some $u, x', y' \in V_h^*$ and $a \neq b$; since $L$ is prefix-free, such $a$ and $b$ exist).

$L$ *represents* a graph $g$ iff $\mathrm{gra}(L)$ **isom** $g$.

Clearly, the intuition behind the graph $h$ in this definition is that it is an alphabet with a graph structure. Thus, a grep language $L$ over $h$ is a language over the (ordinary) alphabet $V_h$, while $E_h$ is used to define the edges of $\mathrm{gra}_h(L)$.

For a graph $h$, we denote by $\mathrm{Rep}(h)$ the set of all graphs $g$ for which there exists a grep language $L$ over $h$ such that $\mathrm{gra}(L)$ **isom** $g$. In other words, $\mathrm{Rep}(h)$ consists of all graphs that are represented by a grep language over $h$.

Every graph $g$ can be represented by a grep language in a trivial way, taking the graph itself as the alphabet graph. In fact, if $h = g$ and $L = V_h$, then $L$ is a grep language over $h$

with $\mathrm{gra}_h(L) = g$; note that $L$ contains strings of length one only.

Whereas a grep language $L$ over $h$ represents a graph, $L$ itself can be represented in several well-known ways. First, since $L$ is prefix-free, it can be represented by a unique rooted directed (unordered) tree $T$ of which the edges are labeled by symbols from the alphabet $V_h$, in such a way that the edges leaving a node of $T$ have distinct labels. $T$ represents $L$ in the sense that $L$ is the set of all label sequences of the paths from the root of $T$ to its leaves. For two such paths, with label sequences $x$ and $y$, the first symbols $a$ and $b$ where $x$ and $y$ differ (as in the definition of $\mathrm{gra}(L)$) are precisely the labels of the first two edges where the paths part (in [EhrRoz2, Definition 5.1] $(a, b)$ is called the branching pair of $x$ and $y$). The nodes of $T$ are in one-to-one correspondence with the prefixes of $L$. In particular, the leaves of $T$ correspond to the elements of $L$ and, hence, to the vertices of $\mathrm{gra}(L)$. Note that for the two leaves corresponding to $x$ and $y$, as above, the symbols $a$ and $b$ label outgoing edges of the least common ancestor of the two leaves.

Second, $L$ may be represented by any finite automaton $A$ that recognizes $L$. Note that $A$ is a directed graph of which the vertices are called states (with an initial state and final states) and of which the edges are labeled by symbols from $V_h$. In particular we may require that $A$ is deterministic and that its final states have no outgoing edges. For two strings $x$ and $y$ from $L$, we can find their first difference $(a, b)$ by following the paths in $A$ corresponding to $x$ and $y$ (which are unique because of the determinism of $A$) and see where they part. Note that the tree $T$ discussed above is such a finite automaton, where the root of $T$ is its initial state and the leaves of $T$ are its final states. This is the deterministic automaton accepting $L$ with the maximal number of (useful) states. Another automaton of interest is the minimal deterministic automaton $A_{\min}$ that accepts $L$; just as the tree $T$, it is a unique representation of $L$. Since $L$ is finite, $A_{\min}$ is acyclic, and since $L$ is nonempty and prefix-free, $A_{\min}$ has exactly one final state. Just as in $T$, the elements of L uniquely correspond to the paths in $A_{\min}$ that lead from the initial state to the final state. It is not difficult to see that $A_{\min}$ can be obtained from the tree $T$ by sharing all equal subtrees of $T$ (see [Eil, Section III.5]). Thus, if $T$ has many equal subtrees, then $L$ and, hence, $\mathrm{gra}(L)$, has a very compact representation $A_{\min}$. We will see later that this means that $\mathrm{gra}(L)$ has a very regular clan structure.

EXAMPLE 2. We give three examples of grep languages. The first example shows that our approach generalizes the usual representation of numbers by positional notation.

(1) Let $h$ be the two-vertex one-edge graph with $V_h = \{0, 1\}$ and $E_h = \{(0, 1)\}$, and let $n \in \mathbb{N}$. Then $L_n = 1 \cdot \{0, 1\}^n$ is a grep language over $h$ (consisting of all bitstrings of length $n+1$, starting with a 1). For $x, y \in L_n$, there is an edge from $x$ to $y$ in $\mathrm{gra}_h(L_n)$ iff the first bit in which $x$ and $y$ differ (counting from the left) is 0 in $x$ and 1 in $y$, i.e., iff the number denoted by $x$ is smaller than the number denoted by $y$. Thus, $\mathrm{gra}(L_n)$ is a linear order with $2^n$ vertices. The graph $\mathrm{gra}(L_2)$ is shown in Fig. 2a; it corresponds to the linear order $100 < 101 < 110 < 111$. The tree $T$ corresponding to $L_2$ is given in Fig. 2b, and the minimal automaton $A_{\min}$ for $L_2$ is given in Fig. 2c, where the initial state is indicated by a double arrow and the final state is encircled. Note that $A_{\min}$ is obtained from $T$ by sharing equal subtrees; this means in particular that all leaves of $T$ are identified. It should be clear that, in general, the minimal automaton $A_{\min}$ for $L_n$ has just $n+2$ vertices and, thus, is a very compact representation of the graph $\mathrm{gra}_h(L_n)$ which has $2^n$ vertices.

(2) Let $h$ be the undirected graph shown in Fig. 3a, where, formally, an undirected edge $\{x, y\}$ is a pair of directed edges $(x, y)$ and $(y, x)$. Consider the grep language $L = \{a, c\} \cdot \{a, b\} \cdot \{a, c\} = \{aaa, aac, aba, abc, caa, cac, cba, cbc\}$ over $h$. The (undirected) graph $\mathrm{gra}(L)$ is given in Fig. 3b; it consists of two squares. The tree corresponding to $L$ is shown in Fig. 3c, and the minimal automaton in Fig. 3d. For the grep language $L' = \{a, b\} \cdot L$, $\mathrm{gra}(L')$ can be obtained by taking two disjoint copies of $\mathrm{gra}(L)$ and by connecting each vertex of the first copy with each vertex of the second copy (by an undirected edge). Thus, $\mathrm{gra}(L')$ has 16 vertices and $16 + 64 = 80$ edges, but, obviously, the corresponding minimal automaton $A_{\min}$ has just 5 vertices and 8 edges; this should convince the reader that (the graph) $A_{\min}$ is a compact representation of a very complicated graph (that has, however, a very regular structure).

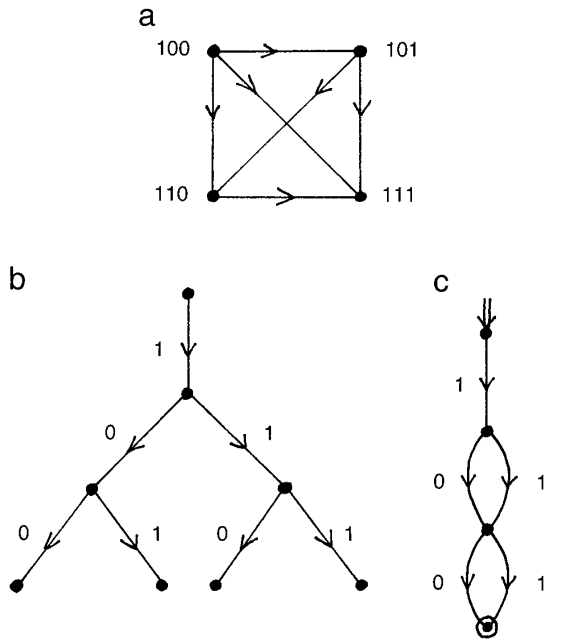


**FIG. 2.** Example 2(1).

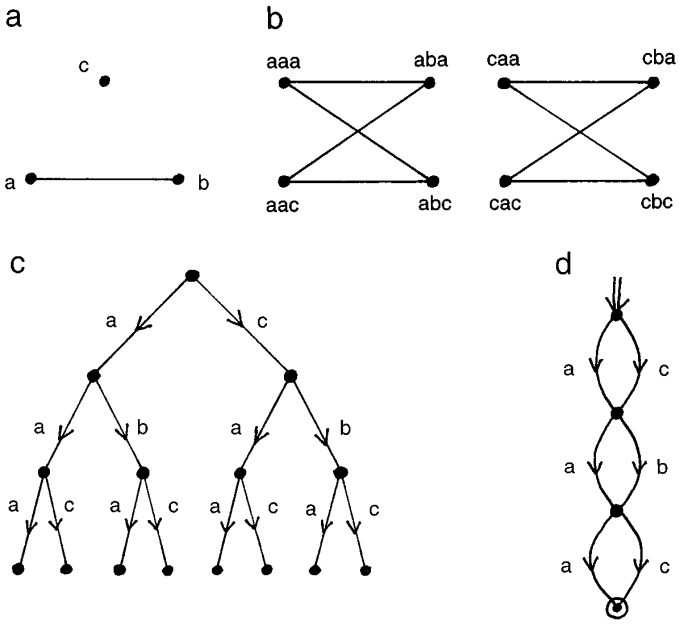


**FIG. 3.** Example 2(2).

(3) Consider finally the graph $h$ in Fig. 4a and the grep language $L = \{bd, bed, bee, caa, cab, daa, dab, db\}$ over $h$. The tree corresponding to $L$ is shown in Fig. 4b, gra($L$) in Fig. 4c, and the minimal automaton for $L$ in Fig. 4d.

We now introduce a notation for the *first difference* of two distinct strings; this notation will be very useful in a number of proofs in the sequel. Let $V$ be an alphabet. Intuitively, we view the strings in $V^*$ as extended with $B$'s, where $B$ is a new symbol, standing for "blank." For $u, v \in V^*$ with $u \neq v$, we define $\text{first}(u, v) \in (V \cup \{B\}) \times (V \cup \{B\})$ recursively as follows (with $a, b \in V$ and $\lambda$ is the empty string):

$$\text{first}(au, \lambda) = (a, B), \quad \text{first}(\lambda, bv) = (B, b),$$

$$\text{first}(au, bv) = \begin{cases} (a, b) & \text{if} \quad a \neq b \\ \text{first}(u, v) & \text{if} \quad a = b. \end{cases}$$

This function has the following elementary properties (with $u, v, v', u_1, u_2, v_1, v_2$ in $V^*$ and $u \neq v$, $u_1 \neq u_2$, and $v_1 \neq v_2$):

(F0) $\text{first}(u, v) \neq (B, B)$,

(F1) $u$ is a prefix of $v$ iff $\text{first}(u, v) = (B, b)$ for some $b \in V$ iff $\text{first}(v, u) = (b, B)$ for some $b \in V$,

(F2) $\text{first}(uv_1, uv_2) = \text{first}(v_1, v_2)$, and

(F3) if $\text{first}(u_1, u_2) \in V \times V$, then $\text{first}(u_1 v, u_2 v') = \text{first}(u_1, u_2)$.

Using this notation we have a compact way of expressing $E_{\text{gra}(L)}$ through $E_h$ and "first" (where $h$ is a graph and $L$ is a grep language over $h$): if $x$ and $y$ are distinct elements of $L$, then $E_{\text{gra}(L)}(x, y) = E_h(\text{first}(x, y))$. Note that (F0) and (F1), together with the prefix-freeness of $L$, imply that $\text{first}(x, y) \in V_h \times V_h$.

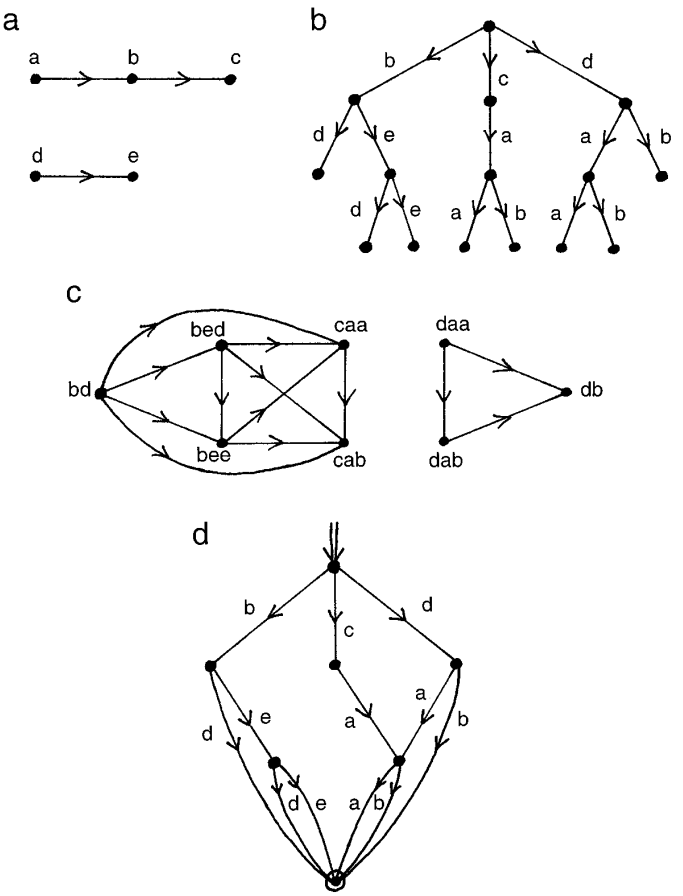


**FIG. 4.** Example 2(3).

As mentioned already in the Introduction, we investigate in this paper how the (formal language theoretic) structure of a grep language $L$ influences the (graph theoretic) structure of the graph gra($L$) that it represents. We start with some easy properties; in particular we consider the classical operation of taking the quotient of a language by a string (see, e.g., [RabSco, Brz, Eil, HopUll]). Let $L$ be a grep language over $h$.

If $L'$ is another grep language over $h$ with $L' \subseteq L$, then gra($L'$) is a subgraph of gra($L$); recall that "subgraph" always means "induced subgraph."

If $u \in V_h^*$, then the languages

$$uL = \{uv \mid v \in L\},$$

$$u/L = \{v \in V_h^* \mid uv \in L\},$$

$$\text{pref}(u, L) = u(u/L) = uV_h^* \cap L$$

are all prefix-free (and, hence, they are grep languages over $h$, provided they are nonempty). For the first two languages prefix-freeness follows from properties (F1) and (F2). The

language $u/L$ is the *quotient* of $L$ by $u$, and $\mathrm{pref}(u, L)$ is the set of all strings in $L$ that have prefix $u$. Note that $u/L$ and $\mathrm{pref}(u, L)$ are nonempty (and, hence, grep languages over $h$) iff $u$ is a prefix of $L$. Let us now consider the graphs defined by these languages. First, by property (F2), $\mathrm{gra}(uL)$ **isom** $\mathrm{gra}(L)$, with the isomorphism $\phi: uL \to L$ such that $\phi(uv) = v$. Then, since $\mathrm{pref}(u, L) \subseteq L$, $\mathrm{gra}(\mathrm{pref}(u, L))$ is a subgraph of $\mathrm{gra}(L)$.

The following lemma expresses the relation between the quotients of a grep language and the clans of the graph it defines. It demonstrates that the notion of clan also naturally arises in a language-theoretic framework.

LEMMA 3. *Let $L$ be a grep language over $h$, and let $u$ be a prefix of $L$:*

(1) $\mathrm{pref}(u, L)$ *is a clan of* $\mathrm{gra}(L)$,

(2) $\mathrm{gra}(u/L)$ **isom** $\mathrm{gra}(\mathrm{pref}(u, L))$, *and, hence,* $\mathrm{gra}(u/L)$ *is isomorphic to a clan of* $\mathrm{gra}(L)$.

*Proof.* (1) Let $x \in L - \mathrm{pref}(u, L)$. Then $u$ is not a prefix of $x$, and, moreover, $x$ is not a prefix of $u$ (because $u$ is a prefix of $L$). Hence, by property (F1), $\mathrm{first}(u, x) \in V_h \times V_h$. And so, by property (F3), $\mathrm{first}(uv, x) = \mathrm{first}(u, x)$ for every $v \in V_h^*$. Similarly, $\mathrm{first}(x, uv) = \mathrm{first}(x, u)$. This shows that $\mathrm{pref}(u, L)$ is a clan of $\mathrm{gra}(L)$.

(2) $\mathrm{gra}(u/L)$ **isom** $\mathrm{gra}(u(u/L)) = \mathrm{gra}(\mathrm{pref}(u, L))$. ∎

As an example, consider the grep language $L$ from Example 2(3) (cf. Fig. 4b). Then $\mathrm{pref}(b, L) = \{bd, bed, bee\}$. It can be seen directly from Fig. 4b that $\mathrm{pref}(b, L)$ is indeed a clan of $\mathrm{gra}(L)$: every string not in $\mathrm{pref}(b, L)$ starts with $c$ or with $d$. It is easy to verify in Fig. 4c that $\{bd, bed, bee\}$ is indeed a clan of $\mathrm{gra}(L)$.

Lemma 3 shows that all (nonempty) *quotients* of $L$ *are clans* of $\mathrm{gra}(L)$. Viewing $L$ as a tree $T$, as discussed before, the nonempty quotients of $L$ are the subtrees of $T$: if the prefix $u$ of $L$ corresponds to the vertex $x$ of $T$, then $u/L$ is represented by the subtree of $T$ rooted at $x$ (and note that $\mathrm{pref}(u, L)$ consists of all label sequences of paths going through $x$). It is also well known (see, e.g., [Eil, Theorem III.5.2]) that the minimal automaton $A_{\min}$ of $L$, also discussed before, can be constructed by taking all nonempty quotients of $L$ as the states of $A_{\min}$; the initial state is then $L$ itself, and the final state is $\{\lambda\}$. Thus, for prefixes $u$ and $v$ of $L$, $u/L = v/L$ iff $u$ and $v$ lead to the same state $x$ of $A_{\min}$ (starting from the initial state), and in that case $u/L\ (= v/L)$ consists of all strings that lead from $x$ to the final state. The equivalence relation $u/L = v/L$ for strings $u, v$ is the well-known right-invariant equivalence relation of Nerode (see, e.g., [Eil, Section III.9]).

Clearly, only certain clans of $\mathrm{gra}_h(L)$ correspond to quotients of $L$, depending on the particular choice of $h$ and $L$. For instance, in Example 2(1), $\{101, 110\}$ is a clan of $\mathrm{gra}(L_2)$, but not a quotient of $L_2$ (in the sense that it is not equal to $\mathrm{pref}(u, L)$ for any $u$). However, it is not difficult to prove that when $h$ and $L$ are allowed to vary, every clan $X$ of a graph $g$ is a quotient, in the sense that there is a representation $\mathrm{gra}_h(L)$ of $g$ such that $\phi(\mathrm{pref}(u, L)) = X$ for some prefix $u$ of $L$, where $\phi$ is the isomorphism from $\mathrm{gra}_h(L)$ to $g$. In fact, take $x_0 \in X$, and let $h = g$ and $L = (V_g - X) \cup x_0 X$. Then it is easy to verify that $\mathrm{gra}_h(L)$ **isom** $g$ with the isomorphism $\phi$ from $\mathrm{gra}_h(L)$ to $g$ defined by: $\phi(y) = y$ for $y \in V_g - X$, and $\phi(x_0 x) = x$ for $x \in X$ (note that $E_g(y, x) = E_g(y, x_0)$ and $E_g(x, y) = E_g(x_0, y)$ for every $y \in V_g - X$ and $x \in X$, because $X$ is a clan). Now, $\phi(\mathrm{pref}(x_0, L)) = \phi(x_0(x_0/L)) = x_0/L = X$.

We now turn to the fundamental relationship between concatenation of (grep) languages and substitution of graphs. The classical operation of concatenation is the concatenation of two languages $L_1$ and $L_2$: every string of $L_1$ is concatenated with every string of $L_2$. Here we consider a more general concatenation operation: the concatenation of a language with a family of languages. To each string $x$ of a language $L$ we associate a language $L_x$, and we concatenate $x$ with every string of $L_x$ (only). The result of this concatenation is the language $\bigcup \{xL_x \mid x \in L\}$. The next theorem shows that this generalized concatenation operation corresponds precisely to graph substitution.

THEOREM 4. *Let $L$ be a grep language over $h$, and let $L_x$ be a grep language over $h$, for every $x \in L$. Then $\bigcup \{xL_x \mid x \in L\}$ is a grep language over $h$,* $\mathrm{gra}(\bigcup \{xL_x \mid x \in L\})$ **isom** $\mathrm{gra}(L)[x \leftarrow \mathrm{gra}(L_x)]_{x \in L}$, *and, in particular, the concatenation function $\phi(x, y) = xy$ is an isomorphism from* $\mathrm{gra}(L)[x \leftarrow \mathrm{gra}(L_x)]_{x \in L}$ *to* $\mathrm{gra}(\bigcup \{xL_x \mid x \in L\})$.

*Proof.* Note that $\bigcup \{xL_x \mid x \in L\} = \{xy \mid x \in L, y \in L_x\}$, and note that the set of vertices of $\mathrm{gra}(L)[x \leftarrow \mathrm{gra}(L_x)]_{x \in L}$ is $\{(x, y) \mid x \in L,\ y \in L_x\}$. To show simultaneously that $\bigcup \{xL_x \mid x \in L\}$ is prefix-free and that the concatenation function $\phi(x, y) = xy$ is an isomorphism from $g = \mathrm{gra}(L)[x \leftarrow \mathrm{gra}(L_x)]_{x \in L}$ to $g' = \mathrm{gra}(\bigcup \{xL_x \mid x \in L\})$, let $x_1, x_2 \in L$, $y_1 \in L_{x_1}$, and $y_2 \in L_{x_2}$. Since $L$ is prefix-free, $x_1 y_1 = x_2 y_2$ implies $x_1 = x_2$ and $y_1 = y_2$. Hence $\phi$ is a bijection. Assume now that $x_1 y_1 \neq x_2 y_2$; then $x_1 \neq x_2$, or $x_1 = x_2$ and $y_1 \neq y_2$. By properties (F1)–(F3),

$$\mathrm{first}(x_1 y_1, x_2 y_2) = \begin{cases} \mathrm{first}(x_1, x_2) & \text{if} \quad x_1 \neq x_2 \\ \mathrm{first}(y_1, y_2) & \text{if} \quad x_1 = x_2, \end{cases}$$

and so $\bigcup \{xL_x \mid x \in L\}$ is prefix-free (by prefix-freeness of $L$ and all $L_x$). Also, $E_{g'}(\phi(x_1, y_1), \phi(x_2, y_2)) = E_g((x_1, y_1), (x_2, y_2))$, which means that $\phi$ is an isomorphism from $g$ to $g'$. ∎

Note that the concrete way in which we have defined substitution (with ordered pairs as vertices) has given us the possibility of using the operation of concatenation as isomorphism, in a natural way.

Note also that for the grep language $L' = \bigcup \{xL_x \mid x \in L\}$ and for any $x \in L$, $x/L' = L_x$. In other words, the languages $L_x$ are quotients of $L'$. This implies that this generalized type of concatenation is also natural from the point of view of the relationship between prefix-free languages and their trees: the tree $T'$ corresponding to the grep language $\bigcup \{xL_x \mid x \in L\}$ is obtained by attaching $T_x$ to the leaf of $T$ that corresponds to $x$, where $T_x$ is the tree of $L_x$ and $T$ is the tree of $L$. It is perhaps interesting to observe that $T$ corresponds to the tree above a cut of $T'$, while the trees $T_x$ correspond to the trees below that cut. It should be clear that, vice versa, for a given grep language, every cut through its tree represents a way of viewing the language as a generalized concatenation.

As an example of Theorem 4, consider again the language $L$ of Example 2(3) (see Fig. 4). Obviously, $L = \bigcup \{xL_x \mid x \in L_0\}$, where $L_0 = \{b, ca, da, db\}$, $L_b = \{d, ed, ee\}$, $L_{ca} = L_{da} = \{a, b\}$, and $L_{db} = \{\lambda\}$. These grep languages correspond to a cut through the tree of $L$ in Fig. 4b, as indicated in Fig. 5a. Let us now consider the graphs represented by these grep languages. The graph $\mathrm{gra}(L_0)$ is shown in Fig. 5b. Since $L_b = b/L$, Lemma 3 implies that $\mathrm{gra}(L_b)$ **isom** $\mathrm{gra}(\mathrm{pref}(b, L)) = \mathrm{gra}(\{bd, bed, bee\})$; from Fig. 4c it can now be seen that $\mathrm{gra}(L_b)$ is the acyclic triangle. Obviously, $\mathrm{gra}(L_{ca})$ and $\mathrm{gra}(L_{da})$ are graphs with one edge and two vertices and $\mathrm{gra}(L_{db})$ is the one-vertex graph. It is now easy to verify from Figs. 4c and 5b that, as proved in Theorem 4, $\mathrm{gra}(L)$ is isomorphic to the result of substituting the graphs $\mathrm{gra}(L_x)$ into the vertices $x$ of $\mathrm{gra}(L_0)$.

Theorem 4 can be viewed as a compositional result, in the sense that it shows that the meaning of a grep language is uniquely determined by the meanings of its components (where the "meaning" of a grep language $L$ is the isomorphism class of the graph $\mathrm{gra}(L)$, and its "components" are its quotients). From the point of view of trees it shows the usual compositionality; the meaning of a tree is determined by the meaning of its subtrees. Thus, the trees may be viewed as expressions denoting graphs, where the expressions are formed with the operation of graph substitution. We will discuss this aspect in more detail in the next section (Theorem 8(2)).

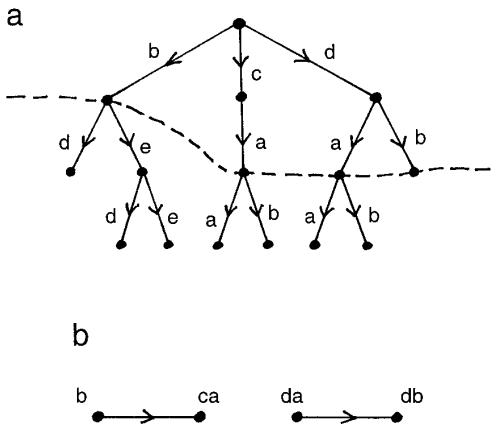


**FIG. 5.** A cut through the tree.

As a consequence of this compositionality we obtain that, in a grep language, we can replace quotients by equivalent quotients (or, subtrees by equivalent subtrees).

LEMMA 5. *Let $L$, $M$ be grep languages over $h$, let $u$ be a prefix of $L$, and let $L' = (L - u(u/L)) \cup uM$. If* $\mathrm{gra}(u/L)$ **isom** $\mathrm{gra}(M)$, *then* $\mathrm{gra}(L)$ **isom** $\mathrm{gra}(L')$.

*Proof.* $L'$ is intuitively the result of replacing the quotient $u/L$ by $M$ in $L$. Note that $\mathrm{pref}(u, L) = u(u/L)$. Consider the grep language $K = (L - \mathrm{pref}(u, L)) \cup \{u\}$. Intuitively, $K$ is the result of removing $u/L$ from $L$ (i.e., removing the subtree corresponding to $u$). For $x \in K$, define the grep languages $L_x$ and $L'_x$ as $L_u = u/L$, $L'_u = M$, and $L_x = L'_x = \{\lambda\}$ for $x \neq u$. Then $\bigcup \{xL_x \mid x \in K\} = L$ and $\bigcup \{xL'_x \mid x \in K\} = L'$. Hence, by Theorem 4, $\mathrm{gra}(L)$ **isom** $\mathrm{gra}(K)[x \leftarrow \mathrm{gra}(L_x)]$ and $\mathrm{gra}(L')$ **isom** $\mathrm{gra}(K)[x \leftarrow \mathrm{gra}(L'_x)]$. Since $\mathrm{gra}(L_x)$ **isom** $\mathrm{gra}(L'_x)$ for all $x \in K$, and the substitution behaves well with respect to isomorphism, $\mathrm{gra}(K)[x \leftarrow \mathrm{gra}(L_x)]$ **isom** $\mathrm{gra}(K)[x \leftarrow \mathrm{gra}(L'_x)]$, and so $\mathrm{gra}(L)$ **isom** $\mathrm{gra}(L')$. ∎

Using this replacement lemma we will show that grep languages can be reduced, in the following sense.

DEFINITION 6. Let $L$ be a grep language over $h$. $L$ is *reduced* if for all prefixes $u$, $v$ of $L$: if $\mathrm{gra}(u/L)$ **isom** $\mathrm{gra}(v/L)$, then $u/L = v/L$.

Intuitively this means that the tree corresponding to $L$ does not have different subtrees that represent isomorphic graphs. It also means that in the minimal automaton $A_{\min}$ corresponding to $L$ there do not exist different states that represent isomorphic graphs. In the next theorem we show that for every grep language $L$ we can find an equivalent one (over the same alphabet) that is reduced. Moreover, in general, the new language has less quotients than $L$. In other words, an "equivalent" automaton can be found with less states than the minimal automaton for $L$. Of course, this does not contradict the minimality of $A_{\min}$ because we now also have a "semantic" criterion for comparing states (two states are "semantically equivalent" if they represent isomorphic graphs).

As an introductory example, consider the grep language $L = \{bd, bed, bee, caa, cab, daa, dab, db\}$ from Example 2(3); cf. Fig. 4. Clearly, $\mathrm{gra}(b/L) = \mathrm{gra}(\{d, ed, ee\})$ and $\mathrm{gra}(d/L) = \mathrm{gra}(\{aa, ab, b\})$ are both isomorphic with the acyclic triangle. Also $\mathrm{gra}(c/L)$, $\mathrm{gra}(ca/L)$, and $\mathrm{gra}(be/L)$ are

all isomorphic with the two-vertex one-edge graph. Thus, $L$ is certainly not reduced. Replacing $d/L$ by $b/L$ and replacing $c/L$ by $be/L$, in the sense of Lemma 5, we obtain the grep language $L' = \{bd, bed, bee, cd, ce, dd, ded, dee\}$ over $h$ such that $\text{gra}(L')$ is isomorphic with $\text{gra}(L)$, where the strings of $L$ and $L'$ are listed in the order corresponding to their isomorphism. It is easy to check that $L'$ is reduced. The minimal automaton for $L'$ is given in Fig. 6a; it has four states, instead of the seven of the minimal automaton for $L$ (in Fig. 4d). It may be amusing to observe that a nonminimal automaton can be "semantically" better than a minimal one; Fig. 6b shows a nonminimal automaton (for $L'$) that represents the same graph (modulo isomorphism) as the minimal automaton (for $L$) of Fig. 4d, but that has less states!

THEOREM 7. *For every grep language $L$ over $h$ there exists a reduced grep language $L'$ over $h$ such that*

(1) $\text{gra}(L')$ **isom** $\text{gra}(L)$, *and*

(2) *for every prefix $u$ of $L'$ there exists a prefix $v$ of $L$ such that* $\text{gra}(u/L')$ **isom** $\text{gra}(v/L)$.

*Proof.* This is essentially a result that holds for arbitrary trees (or expressions): if one can replace a subtree by another subtree with the same "meaning" (as we know in this case from Lemma 5), then one can always find an "equivalent" tree such that different subtrees have different meanings. Thus the following proof can be best understood by viewing the grep languages as trees.

For an arbitrary grep language $L$ over $h$, define $\text{quo}(L) = \{u/L \mid u \text{ is a prefix of } L\}$. Intuitively, $\text{quo}(L)$ is the set of all subtrees of the tree $L$. For a family $X$ of such grep languages, let $\text{quo}(X) = \bigcup \{\text{quo}(L) \mid L \in X\}$. We say that $X$ is quotient-closed if $\text{quo}(X) \subseteq X$; intuitively, $X$ is a set of trees that is closed under taking subtrees. Note that $\text{quo}(\text{quo}(L)) \subseteq \text{quo}(L)$ because $v/(u/L) = (uv)/L$. Note also that $L \in \text{quo}(L)$ because $\lambda/L = L$.

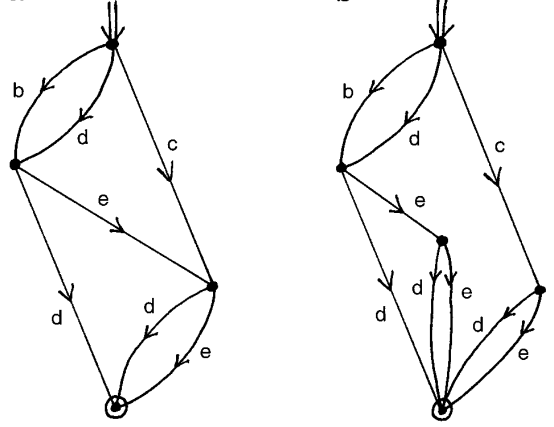


**FIG. 6.** Reduced automata.

Let $L_0$ be an arbitrary grep language. We will show the existence of a reduced grep language $L_0'$ satisfying the requirements of the theorem. We now claim that the following statement holds:

For every quotient-closed $X \subseteq \text{quo}(L_0)$ there is a mapping $\tau: X \to \mathscr{L}(h)$, where $\mathscr{L}(h)$ is the family of all grep languages over $h$, such that

(1) for every $L \in X$, $(\text{gra}(\tau(L))$ **isom** $\text{gra}(L)$,

(2) for all $L_1, L_2 \in X$ and all $L_1' \in \text{quo}(\tau(L_1))$ and $L_2' \in \text{quo}(\tau(L_2))$, if $\text{gra}(L_1')$ **isom** $\text{gra}(L_2')$, then $L_1' = L_2'$, and

(3) for every $L \in X$ and every $M \in \text{quo}(\tau(L))$ there exists $M' \in \text{quo}(L_0)$ such that $\text{gra}(M)$ **isom** $\text{gra}(M')$.

It is easy to see that taking $X = \text{quo}(L_0)$ and $L_0' = \tau(L_0)$ then proves the theorem. So it remains to prove the above statement. We do this by induction on the cardinality of $X$. For $X = \varnothing$ there is nothing to prove. Now consider a nonempty quotient-closed subset $X$ of $\text{quo}(L_0)$. Let $L$ be an element of $X$ of maximal depth, where the depth of $L$ is the maximal length of its strings (this is just the depth of the tree corresponding to $L$). Then $X - \{L\}$ is quotient-closed. Hence, by induction, there is a mapping $\tau: X - \{L\} \to \mathscr{L}(h)$ that satisfies (1)–(3). We extend $\tau$ to $X$ by defining $\tau(L)$ as follows, where we distinguish two cases.

*Case* (i). There exist $L' \in X - \{L\}$ and $M \in \text{quo}(\tau(L'))$ such that $\text{gra}(M)$ **isom** $\text{gra}(L)$. Then we define $\tau(L) = M$. Obviously (1)–(3) still hold for the extended $\tau$.

*Case* (ii). Case (i) does not hold. Let $\{a_1/L, ..., a_k/L\} = \{a/L \mid a \in V_h\} - \{\varnothing\}$. Intuitively, this is the set of all direct subtrees of the tree $L$. Clearly, $a_i/L \in X - \{L\}$ for all $1 \leqslant i \leqslant k$. We define $\tau(L)$ to be the result of replacing $a_i/L$ by $\tau(a_i/L)$ in $L$ for $1 \leqslant i \leqslant k$, in the sense of Lemma 5. Thus, $\tau(L) = \bigcup \{a_i \cdot \tau(a_i/L) \mid 1 \leqslant i \leqslant k\}$. By $k$ applications of Lemma 5, $\text{gra}(\tau(L))$ **isom** $\text{gra}(L)$. This shows property (1). For property (2) it suffices to consider the case that $L_1' = \tau(L)$. But clearly the property is true because Case (i) does not hold: $\text{gra}(\tau(L))$ **isom** $\text{gra}(L)$ and, hence, $\text{gra}(\tau(L))$ is not isomorphic to the graph defined by any quotient of any $\tau(L_2)$. For property (3) it also suffices to consider $M = \tau(L)$. Then $\text{gra}(M)$ **isom** $\text{gra}(L)$ and $L \in \text{quo}(L_0)$. This proves the statement and the theorem. ∎

Note that the second property of $L'$, saying that every $\text{gra}(u/L')$ is already isomorphic to some $\text{gra}(u/L)$, implies that $L'$ does not have more quotients than $L$, i.e., $\#\{u/L' | u \text{ is a prefix of } L'\} \leqslant \#\{u/L \mid u \text{ is a prefix of } L\}$. This is seen as follows. Let $[g]$ denote the isomorphism class of a graph $g$. Then the property says that $\{[\text{gra}(u/L')] \mid u \text{ is a prefix of } L'\} \subseteq \{[\text{gra}(u/L)] \mid u \text{ is a prefix of } L\}$. Now, since $L'$ is reduced, $\#\{u/L' \mid u \text{ is a prefix of } L'\} = \#\{[\text{gra}(u/L')] \mid u \text{ is a prefix of } L'\}$. And obviously $\#\{[\text{gra}(u/L')] \mid u \text{ is a prefix of } L'\} \leqslant \#\{[\text{gra}(u/L)] \mid u \text{ is a prefix of } L\} \leqslant \#\{u/L \mid u \text{ is a}$

prefix of $L\}$. Hence the minimal automation for $L'$ has at most the number of states of the one for $L$.

## 3. CHARACTERIZATIONS OF THE CLASS OF REPRESENTABLE GRAPHS

In this section we present two characterizations of the class Rep($h$) of all graphs that can be represented by a grep language over a given (graph) alphabet $h$. The first characterization (that has three variations) shows how the graphs of Rep($h$) can be obtained from certain elementary graphs by the operation of substitution. The second characterization is in terms of subgraphs that have trivial clans only. These characterizations will show in which sense Rep($h$) generalizes the set of cographs [CorLerSte] and the set of transitive VSP graphs [ValTarLaw]. We start with the first characterization. To state it we need some terminology.

Let $G$ be a set of graphs. $G$ is *closed under substitution* if the following holds: if $g \in G$, $g_x \in G$ for every $x \in V_g$, and $g'$ **isom** $g[x \leftarrow g_x]$, then $g' \in G$. Let $h$ be a graph. $G$ is *closed under substitution into subgraphs of h* if the following holds: if $g$ is a subgraph of $h$, $g_x \in G$ for every $x \in V_g$, and $g'$ **isom** $g[x \leftarrow g_x]$, then $g' \in G$. $G$ is *closed under substitution by subgraphs of h* if the following holds: if $g \in G$, $g_x$ is a subgraph of $h$ for every $x \in V_g$, and $g'$ **isom** $g[x \leftarrow g_x]$, then $g' \in G$. Finally, $G$ *contains the one-vertex graph* if it contains all one-vertex graphs.

THEOREM 8. *Let h be a graph*:

(1) Rep($h$) *is the smallest class of graphs that contains all graphs that are isomorphic to a subgraph of h and is closed under substitution.*

(2) Rep($h$) *is the smallest class of graphs that contains the one-vertex graph and is closed under substitution into subgraphs of h.*

(3) Rep($h$) *is the smallest class of graphs that contains the one-vertex graph and is closed under substitution by subgraphs of h.*

*Proof.* We first show one half of (1). By definition, Rep($h$) is closed under isomorphisms. To prove that Rep($h$) contains all subgraphs of $h$, let $X \subseteq V_h$. Then the subgraph $h[X]$ of $h$ is obviously represented by the grep language $X$ over $h$. Note that this also shows that Rep($h$) contains the one-vertex graph. To prove that Rep($h$) is closed under substitution, assume that $L$ represents $g$ and that $L_x$ represents $g_x$ for every $x \in V_g$. Let $\phi$ be the isomorphism from gra($L$) to $g$. Then gra($\bigcup \{xL_{\phi(x)} \mid x \in L\}$) **isom** gra($L$)$[x \leftarrow \text{gra}(L_{\phi(x)})]$ by Theorem 4, and hence the grep language $\bigcup \{xL_{\phi(x)} \mid x \in L\}$ represents $g[\phi(x) \leftarrow g_{\phi(x)}] = g[x \leftarrow g_x]$.

It now remains to show that for each grep language $L$ over $h$ the graph gra($L$) can be obtained (modulo isomorphism) from the one-vertex graph by repeated substitution into, or by, subgraphs of $h$. We prove this by induction on the tree size of $L$, i.e., on the cardinality of the set of all prefixes of $L$. If $L$ has tree size 1, then $L = \{\lambda\}$ and gra($L$) is the one-vertex graph. Now assume that $L$ has tree size $>1$. We first prove the "into" part. Let $X = \{a \in V_h \mid a \text{ is a prefix of } L\}$. Since $L = \bigcup \{a(a/L) \mid a \in X\}$, Theorem 4 implies that gra($L$) **isom** gra($X$)$[a \leftarrow \text{gra}(a/L)]$. Now, $a/L$ has a smaller tree size than $L$. Hence, by induction, gra($a/L$) can be obtained from the one-vertex graph by repeated substitution into subgraphs of $h$. The same is therefore true for gra($L$), because gra($X$) $= h[X]$. Next we show the "by" part. There exists a prefix $u$ of $L$ such that $u/L \subseteq V_h$ ($u$ corresponds to a vertex of the tree of which all children are leaves). Consider the grep language $K = (L - \text{pref}(u, L)) \cup \{u\}$, as in the proof of Lemma 5. For $x \in K$, define the grep languages $L_x$ as follows (also as in the proof of Lemma 5): $L_u = u/L$ and $L_x = \{\lambda\}$ for $x \neq u$. Then $L = \bigcup \{xL_x \mid x \in K\}$. Hence, by Theorem 4, gra($L$) **isom** gra($K$)$[x \leftarrow \text{gra}(L_x)]$. Since $K$ has smaller tree size than $L$, gra($K$) can be obtained from the one-vertex graph by repeated substitution by subgraphs of $h$. This proves that the same is true for gra($L$), because both gra($L_u$) $= h[u/L]$ and the one-vertex graphs gra($L_x$) $=$ gra($\{\lambda\}$) are subgraphs of $h$. ∎

As an example of Theorem 8(2), we consider again the grep language $L$ over $h$ from Example 2(3) (cf. Fig. 4) and verify that gra($L$) can be obtained from the one-vertex graph by repeated substitution into subgraphs of $h$. First, gra($be/L$) $= h[\{d, e\}]$, which is isomorphic to the graph obtained by substituting the one-vertex graph into both vertices of the subgraph $h[\{d, e\}]$ of $h$. Second, gra($b/L$) is isomorphic to $(h[\{d, e\}])[x \leftarrow g_x]$, where $g_d$ is the one-vertex graph and $g_e = h[\{d, e\}]$. Hence, gra($b/L$) is isomorphic to the acyclic triangle (as we have observed before). Finally, it can be seen from Fig. 4c that gra($L$) is isomorphic to the graph $(h[\{b, c, d\}])[x \leftarrow g_x]$, where $g_b$ and $g_d$ are both the acyclic triangle, and $g_c$ **isom** $h[\{d, e\}]$. This shows that, indeed, gra($L$) can be obtained by repeated substitution into subgraphs of $h$.

Theorem 8(2) is another way of expressing the compositionality theorem (Theorem 4). It says that, modulo isomorphism, the graphs of Rep($h$) can be denoted by expressions that are formed with one constant, denoting the one-vertex graph, and as many operators as there are subgraphs of $h$, each such operator denoting the graph operation of substitution into the corresponding subgraph of $h$. These expressions are in fact in one-to-one correspondence with (the trees corresponding to) the grep languages over $h$. As an example, let $o$ be the constant that denotes the one-vertex graph and let $\sigma_{v_1 \cdots v_k}$ be the operator that denotes the operation of substitution into $h[\{v_1, ..., v_k\}]$, i.e., to be precise, $\sigma_{v_1 \cdots v_k}(g_1, ..., g_k) = (h[\{v_1, ..., v_k\}])[x \leftarrow g_x]$, where

$g_{v_i} = g_i$ for $1 \leqslant i \leqslant k$. Then, as discussed above for Example 2(3), gra($L$) is isomorphic to $\sigma_{bcd}(\sigma_{de}(o, \sigma_{de}(o, o)), \sigma_a(\sigma_{ab}(o, o)), \sigma_{ab}(\sigma_{ab}(o, o), o))$. The usual picture of this expression as a labeled ordered tree is given in Fig. 7. Clearly, it is in one-to-one correspondence with the tree of Fig. 4b, in the way discussed in the Introduction; rather than labeling a vertex of the tree by an operator $\sigma_{v_1 \cdots v_k}$ (and ordering its $k$ children), one can equivalently label the $k$ outgoing edges of the vertex by the selectors $v_1, ..., v_k$.

Theorem 8(3) is essentially the same as [EhrRoz2, Theorem 7.11]. As observed there, substitution by a subgraph of $h$ can be viewed as a rewriting step of a context-free graph grammar; the tree corresponding to a grep language $L$ can be viewed as the derivation tree of the derivation of gra($L$) in this graph grammar. This establishes a bridge between the subject of this paper and the area of graph grammars. In fact, to be more precise, it is not difficult to show that, for every graph $h$, the set of graphs Rep($h$) can be generated by a so-called neighbourhood-uniform NLC graph grammar (see [JanRoz]). This implies that Rep($h$) can also be generated by a C-edNCE graph grammar, which is a more powerful type of graph grammar (see, e.g., [EngRoz]). It is shown in [Eng] that every set of graphs that is generated by a C-edNCE graph grammar, can be represented by a set of trees in a way that naturally generalizes the representation method in this paper.

Theorem 8 relates arbitrary substitutions (1), bottom-up substitutions (2), and top-down substitutions (3). Equality of the three classes can also be shown alternatively in a direct way (i.e., without the use of Rep($h$)) through the *associativity of graph substitution* (cf. [Cou1]), which means the following. Let $g$ be a graph, let $g_x$ be a graph for every $x \in V_g$, and let $g_{xy}$ be a graph for every $x \in V_g$ and $y \in V_{g_x}$; then $g[x \leftarrow g_x[y \leftarrow g_{xy}]]$**isom** $(g[x \leftarrow g_x])[(x, y) \leftarrow g_{xy}]$. However, associativity of graph substitution is an immediate consequence of Theorem 4 and the associativity of string concatenation. Thus, our proof of Theorem 8 is essentially the same as the alternative proof suggested above. This means that the associativity of graph substitution is indeed at the basis of our approach.

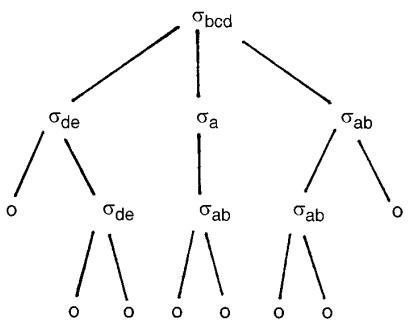


**FIG. 7.** An expression tree.

We now turn to the second characterization. A graph is *primitive* if all its clans are trivial (i.e., consist of all vertices or of exactly one vertex). In other words, a graph is primitive if it cannot be written as a substitution $g[x \leftarrow g_x]$ in a nontrivial way (see, e.g., Remark 5.9 of [Gol]); in [MöhRad] such graphs are said to be prime. We show that, for a graph $h$, the primitive subgraphs of $h$ are the only primitive subgraphs of graphs in Rep($h$). Thus, giving the alphabet graph $h$ amounts to an explicit specification of the primitivity that is allowed in the graphs of Rep($h$).

THEOREM 9. *Let $h$ be a graph. Then* Rep($h$) *is the set of all graphs $g$ such that every primitive subgraph of $g$ is isomorphic to a subgraph of $h$.*

*Proof.* We first show that for every graph $g \in$ Rep($h$) all primitive subgraphs of $g$ are isomorphic to subgraphs of $h$. This is similar to the proof of [EhrRoz2, Theorem 4.4]. Let $g =$ gra($L$) for some grep language $L$ over $h$, and let $X \subseteq L$ be such that $g[X]$ is primitive. Let $u/L$ be the quotient of $L$ with the smallest cardinality such that $X \subseteq$ pref$(u, L) = u(u/L)$; intuitively, $u$ corresponds to the least ancestor of the leaves that correspond to $X$. Consider any $a \in V_h$ such that $ua$ is a prefix of $L$. Then $X$ is not contained in pref$(ua, L)$. Since pref$(ua, L)$ is a clan of $g$ (see Lemma 3), it is easy to see that $X \cap$ pref$(ua, L)$ is either empty or a clan of $g[X]$. Hence, because all clans of $g[X]$ are trivial, $X \cap$ pref$(ua, L)$ is either empty or a singleton. Let $Y = \{a \in V_h \mid X \cap \text{pref}(ua, L) \text{ is a singleton}\} = \{a_1, ..., a_k\}$. Then $X = \{ua_1 v_1, ..., ua_k v_k\}$ for certain $v_1, ..., v_k \in V_h^*$. Since $g[X] =$ gra($X$), this shows that $g[X]$ is isomorphic to the subgraph $h[Y]$ of $h$.

Next we show that all graphs $g$ such that every primitive subgraph of $g$ is isomorphic to a subgraph of $h$, are in Rep($h$). We do this by induction on the cardinality of $V_g$. If $\# V_g = 1$, then $g$ is the one-vertex graph, which is in Rep($h$) by Theorem 8(2). Now assume that $\# V_g > 1$. Let $P = \{P_1, ..., P_k\}$ be a partition of $V_g$ into proper clans (i.e., clans $P_i \neq V_g$) that is maximal, in the sense that there is no such partition coarser than $P$. Note that such a $P$ exists because there is at least one partition of $V_g$ into proper clans, viz. all singleton subsets of $V_g$. Choose $x_i \in P_i$, for $1 \leqslant i \leqslant k$. Let $g' = g/P$ be the quotient graph, i.e., $V_{g'} = P$, and, for distinct $P_i, P_j \in P$, $E_{g'}(P_i, P_j) = E_g(x_i, x_j)$. Note that $g'$ does not depend on the choice of the $x_i$. Then it is easy to see that $g$ **isom** $(g/P)[P_i \leftarrow g[P_i]]_{1 \leqslant i \leqslant k}$. Due to the maximality of $P$, $g/P$ is primitive. Moreover, $g/P$ is isomorphic to $g[\{x_1, ..., x_k\}]$, and hence $g/P$ is isomorphic to a subgraph $h'$ of $h$. By induction, $g[P_i]$ is in Rep($h$). Hence $g$ is isomorphic to some $h'[x \leftarrow g_x]$, where $g_x$ is in Rep($h$). Theorem 8(2) now implies that $g$ is in Rep($h$). ∎

For a given graph $g \in$ Rep($h$) there are in general several grep languages $L$ over $h$ that represent $g$. We leave it as a further topic of investigation to find an efficient algorithm

that constructs the language $L$ with the smallest minimal automaton, i.e., the most compact representation of $g$. Note that the algorithm of Theorem 7 does not necessarily produce the most compact representation. The second part of the proof of Theorem 9 gives a suggestion for such an algorithm. In general, there are several possible choices for a maximal partition $P = \{P_1, ..., P_k\}$ of $V_g$ into proper clans. A possible strategy is to take $P$ in such a way that as many of the clans $P_i$ as possible are isomorphic. This then leads to a language $L$ with few quotients. For instance, for the graph $g = \mathrm{gra}_h(L_n)$ of Example 2(1), any partition $P = \{P_1, P_2\}$ such that $x_1 < x_2$ for all $x_1 \in P_1$ and $x_2 \in P_2$, is a maximal partition of $V_g$ into proper clans. However, $g[P_1]$ and $g[P_2]$ are only isomorphic in the (unique) case that $P_1$ and $P_2$ are of equal cardinality. This choice leads (recursively) to the compact representation $L = \{0, 1\}^n$ which obviously is the most compact one, with a minimal automaton $A_{\min}$ of $n+1$ states (note that the leading 1 of $L_n = 1 \cdot \{0, 1\}^n$ was only added to stress the similarity with binary number notation).

The second part of the proof of Theorem 9 also shows that in Theorem 8(2) the substitution may be restricted to substitution into primitive subgraphs of $h$ with at least two vertices (note that $g/P$ has at least two vertices). This implies (cf. the first part of the proof of Theorem 8) that we may restrict ourselves to grep languages $L$ over $h$ such that, for every prefix $u$ of $L$, if $X = \{a \in V_h \mid ua \text{ is a prefix of } L\}$, then $h[X]$ is primitive and $\# X \geqslant 2$. This again implies the following corollary.

COROLLARY 10. *Theorem* 8 *holds with* "subgraph($s$) *of* $h$" *replaced by* "primitive subgraph($s$) *of* $h$." *Moreover, in Theorem* 8(2,3) "*subgraphs of* $h$" *may even be replaced by* "*primitive subgraphs of* $h$ *with at least two vertices*."

Note that all graphs with one or two vertices are primitive.

We are now able to explain precisely in which sense we generalize cographs [CorLerSte] and transitive VSP graphs [ValTarLaw]. Let $h_{\mathrm{co}}$ be the graph with $V_{h_{\mathrm{co}}} = \{a, b, c\}$ and $E_{h_{\mathrm{co}}} = \{(a, b), (b, a)\}$. Since we view two directed edges in opposite directions as one undirected edge, $h_{\mathrm{co}}$ is the undirected graph with three vertices and one edge (see Fig. 3a). Clearly, $h_{\mathrm{co}}$ has exactly two nonisomorphic primitive subgraphs with at least two vertices: $h_{\mathrm{co}}[\{a, c\}]$ and $h_{\mathrm{co}}[\{a, b\}]$ (the discrete graph with two vertices and the graph with two vertices and one undirected edge). It should also be clear that substitution of graphs $g_1$ and $g_2$ in the first graph gives the disjoint union of $g_1$ and $g_2$, whereas the substitution of $g_1$ and $g_2$ in the second graph gives the "complete connection" of $g_1$ and $g_2$ (i.e., the result of adding all edges between $g_1$ and $g_2$ in the disjoint union of $g_1$ and $g_2$). Hence, by Theorem 8(2) in the version of Corollary 10, $\mathrm{Rep}(h_{\mathrm{co}})$ is the smallest class of graphs that contains the one-vertex graph and is closed under disjoint union and complete connection. Thus, by the definition of cograph, $\mathrm{Rep}(h_{\mathrm{co}})$ is the set of cographs. An example of a cograph is given in Example 2(2) and Fig. 3b. As observed above (just before Corollary 10), we may assume that the trees corresponding to grep languages over $h_{\mathrm{co}}$ are binary, and the edges leaving a vertex of the tree are labeled either by $a$ and $c$ (corresponding to disjoint union) or by $a$ and $b$ (corresponding to complete connection); see Fig. 3c for an example. These trees are in obvious correspondence with the cotrees that represent cographs. Note that Theorem 8(1) and Corollary 10 say that the set of cographs is the smallest substitution-closed set of graphs containing all one- and two-vertex undirected graphs. Note also that Theorem 9 says that an undirected graph is a cograph iff all its primitive subgraphs have at most two vertices (as shown in [EngHarProRoz, Theorem 3.18]).

As a similar example, let $h_{\mathrm{tsp}}$ be the graph with $V_{h_{\mathrm{tsp}}} = \{a, b, c\}$ and $E_{h_{\mathrm{tsp}}} = \{(a, b)\}$. Then, again by Theorem 8(2) and Corollary 10, $\mathrm{Rep}(h_{\mathrm{tsp}})$ is the smallest class of graphs containing the one-vertex graph and closed under disjoint union and "directed complete connection" (in which all directed edges from $g_1$ to $g_2$ are added to the disjoint union of $g_1$ and $g_2$). Thus, $\mathrm{Rep}(h_{\mathrm{tsp}})$ is the set of TSP graphs [CorLerSte], which is the set of transitive VSP graphs [ValTarLaw]. Clearly, the transitive VSP graphs are in one–one correspondence with the intransitive VSP graphs (or MVSP graphs); thus, the binary trees corresponding to the grep languages over $h_{\mathrm{tsp}}$ are exactly the binary decomposition trees of MVSP graphs [ValTarLaw].

Consider now the graph $h_{\mathrm{unp}}$ with $V_{h_{\mathrm{unp}}} = \{a, b, c, d\}$ and $E_{h_{\mathrm{unp}}} = \{(a, b), (c, d), (d, c)\}$. Then, by Theorem 9, $\mathrm{Rep}(h_{\mathrm{unp}})$ is the set of all graphs that have no primitive subgraphs with more than two vertices; this is the set of uniformly nonprimitive graphs considered in [EngHarProRoz]. It naturally contains all cographs and all TSP graphs; it is the smallest class of graphs containing the one-vertex graph and closed under disjoint union, complete connection, and directed complete connection (see [EngHarProRoz, Theorem 3.26]).

If $h$ is the discrete graph with two vertices, then $\mathrm{Rep}(h)$ is the set of all discrete graphs. If $h$ is the graph with two vertices and one edge, then $\mathrm{Rep}(h)$ is the set of all linear orders (see Example 2(1) and Fig. 2 for an example). If $h$ is the graph with two vertices and two edges, then $\mathrm{Rep}(h)$ is the set of all complete graphs.

As a last example, consider the graph $h$ with $V_h = \{a, b, c\}$ and $E_h = \{(a, b), (b, c)\}$. Since $h$ is primitive, it follows again from Theorem 8(2) and Corollary 10 that $\mathrm{Rep}(h)$ is the smallest set of graphs containing the one-vertex graph and closed under disjoint union, directed complete connection, and "double" directed complete connection (which for graphs $g_1$, $g_2$, $g_3$ results in their disjoint union to which all edges from $g_1$ to $g_2$ and all edges from $g_2$ to $g_3$ are added).

We end this section by mentioning some other consequences of Theorem 9. First, it is decidable for arbitrary graphs $h_1$ and $h_2$, whether or not $\mathrm{Rep}(h_1) \subseteq \mathrm{Rep}(h_2)$. In fact, $\mathrm{Rep}(h_1) \subseteq \mathrm{Rep}(h_2)$ iff $h_1 \in \mathrm{Rep}(h_2)$ iff every primitive subgraph of $h_1$ is isomorphic to a subgraph of $h_2$.

Second, there is a forbidden subgraph characterization of $\mathrm{Rep}(h)$. Let $N$ be the maximal size of a primitive subgraph of $h$, and let $G_h$ be the (finite) set of all (isomorphism classes of) primitive graphs of size $\leqslant N+2$ that are not a subgraph of $h$. It is shown in [EhrRoz3] that every primitive graph of size $k$ has a primitive subgraph of size $k-1$ or $k-2$. Consequently, $g \in \mathrm{Rep}(h)$ iff $g$ has no subgraph in $G_h$. Note that we may in fact restrict $G_h$ to the elements that are minimal with respect to the subgraph relation. As an example, for $h = h_{\mathrm{unp}}$, $G_h$ has eight minimal elements; these are the forbidden subgraphs presented in [EngHarProRoz, Theorem 3.14].

Third, as shown in [EngHarProRoz, Section 4.3], every graph property that can be defined in monadic second-order logic (with quantification of sets of vertices), is in LOG(CFL) for the graphs of $\mathrm{Rep}(h)$. Note also that the forbidden subgraph characterization of $\mathrm{Rep}(h)$ implies that its membership problem is in DLOG (in fact, for a fixed graph $g_0$, it can be checked in deterministic logarithmic space whether or not a given graph $g$ has a subgraph isomorphic to $g_0$, by systematically searching through all subgraphs of $g$ of the same size as $g_0$). Thus, for fixed $h$, efficient algorithms exist to recognize the graphs in $\mathrm{Rep}(h)$ and to verify several of their properties.

Finally, we mention that Theorem 9 also holds conversely, in the following sense. Let $G$ be a finite set of primitive graphs, and let $\mathrm{PR}(G)$ be the set of all graphs $g$ such that every primitive subgraph of $g$ is isomorphic to a graph in $G$. Then there exists a graph $h$ such that $\mathrm{PR}(G) = \mathrm{Rep}(h)$. In fact, it is easy to see that we may remove from $G$ all graphs which contain a primitive subgraph that is not in $G$. This shows that we may assume that $G$ is subgraph closed. Let $G_{\max}$ be the set of graphs in $G$ that are maximal with respect to the subgraph relation in $G$. We now take $h$ to be the disjoint union of all graphs in $G_{\max}$. Obviously, $G$ equals the set of primitive subgraphs of this $h$.

Theorem 9, together with its converse show that the effect of the alphabet $h$ is to restrict the possible types of primitive subgraphs: for any class $X$ of graphs, there exists a graph $h$ such that $X = \mathrm{Rep}(h)$ iff there exists a finite set of primitive graphs $G$ such that $X = \mathrm{PR}(G)$.

## 4. CONCATENATION AND UNIFORM SUBSTITUTION

In Theorem 4 we have shown the fundamental relationship between a generalized type of concatenation of languages and substitution of graphs. In this section we briefly consider this relationship for the usual type of concatenation. Suppose that a graph $g$ is represented by a grep language $L_0$, and suppose that $L_0$ is the concatenation $L_1 L_2$ of two other grep languages (where $L_1 L_2 = \{xy \mid x \in L_1,\ y \in L_2\}$ as usual). It is easy to see what this means for the structure of $g$ with respect to substitution: it is the case of Theorem 4 with $L_x = L_2$ for every $x \in L_1$. The corresponding substitution of graphs will be called "uniform substitution" (it is called composition or lexicographic product in [Har, Sab2]).

DEFINITION 11. Let $g_1 = (V_1, E_1)$ and $g_2 = (V_2, E_2)$ be graphs. The *uniform substitution* of $g_2$ into $g_1$, denoted $g_1 \leftarrow g_2$, is the graph $g_1[x \leftarrow g_2]_{x \in V_1}$. Thus, it is the graph $(V, E)$ with $V = V_1 \times V_2$, and for distinct $(x_1, y_1)$, $(x_2, y_2) \in V$,

$$E((x_1, y_1), (x_2, y_2)) = \begin{cases} E_1(x_1, x_2) & \text{if } x_1 \neq x_2 \\ E_2(y_1, y_2) & \text{if } x_1 = x_2. \end{cases}$$

For $x \in V_1$, the *x-row* of $g_1 \leftarrow g_2$, denoted $(g_1 \leftarrow g_2)(x, *)$, is the subgraph of $g_1 \leftarrow g_2$ induced by $\{(x, y) \mid y \in V_2\}$. Similarly, for $y \in V_2$, the *y-column* of $g_1 \leftarrow g_2$, denoted $(g_1 \leftarrow g_2)(*, y)$, is the subgraph of $g_1 \leftarrow g_2$ induced by $\{(x, y) \mid x \in V_1\}$.

Examples of uniform substitution are the following: the graph of Fig. 2a is (isomorphic to) the graph $g \leftarrow g$, where $g$ is the two-vertex one-edge graph; the graph of Fig. 3b is $g_1 \leftarrow g_2$, where $g_1$ is the discrete two-vertex graph and $g_2$ is the square.

The row/column terminology in Definition 11 comes, of course, from viewing $g_1 \leftarrow g_2$ as a matrix. As with arbitrary substitution, it should be clear that for every $x \in V_1$, the $x$-row is a clan of $g_1 \leftarrow g_2$ that is isomorphic to $g_2$ (with the second projection $V_1 \times V_2 \to V_2$ as isomorphism). Also, for every $y \in V_2$, the $y$-column of $g_1 \leftarrow g_2$ is isomorphic to $g_1$ (but not necessarily a clan). Thus, the rows of $g_1 \leftarrow g_2$ are a partition of $g_1 \leftarrow g_2$ into isomorphic clans, and the columns are a partition into isomorphic subgraphs. This makes $g_1 \leftarrow g_2$ a graph with a very regular structure.

Observe that the operation of uniform substitution behaves well with respect to isomorphism: if $g_1'$ **isom** $g_1$ and $g_2'$ **isom** $g_2$, then $g_1' \leftarrow g_2'$ **isom** $g_1 \leftarrow g_2$. In fact, if $\phi_i : g_i' \to g_i$ is an isomorphism, then $\langle \phi_1, \phi_2 \rangle : (g_1' \leftarrow g_2') \to (g_1 \leftarrow g_2)$ is an isomorphism, where

$$\langle \phi_1, \phi_2 \rangle (x, y) = (\phi_1(x), \phi_2(y)).$$

The relationship between (ordinary) concatenation of grep languages and uniform substitution of graphs is stated in the next result.

THEOREM 12. *Let $L_1$ and $L_2$ be grep languages over $h$:*

(1) $\mathrm{gra}(L_1 L_2)$ **isom** $(\mathrm{gra}(L_1) \leftarrow \mathrm{gra}(L_2))$, *and in particular the concatenation function $\phi(x, y) = xy$ is an isomorphism from* $\mathrm{gra}(L_1) \leftarrow \mathrm{gra}(L_2)$ *to* $\mathrm{gra}(L_1 L_2)$.

(2) *If $L_1$ and $L_2$ represent graphs $g_1$ and $g_2$ respectively, then $L_1L_2$ represents $g_1 \leftarrow g_2$.*

*Proof.* (1) follows from Theorem 4, and (1) implies (2). ∎

Note that the minimal automaton for $L_1L_2$ can be obtained from those for $L_1$ and $L_2$ by identifying the final state of the one for $L_1$ with the initial state of the one for $L_2$.

As an example, the language $L$ of Example 2(2) equals $L_1L_2$ with $L_1 = \{a, c\}$ and $L_2 = \{a, b\} \cdot \{a, c\}$; cf. Fig. 3. Applying Theorem 12 to $L_2$, we obtain that $\mathrm{gra}(L_2)$ is isomorphic to $\mathrm{gra}(\{a, b\}) \leftarrow \mathrm{gra}(\{a, c\})$, where $\mathrm{gra}(\{a, b\})$ is the two-vertex one-edge graph and $\mathrm{gra}(\{a, c\})$ is the discrete two-vertex graph; in other words, $\mathrm{gra}(L_2)$ is isomorphic to the square. Applying Theorem 12 now to $L$, we obtain that $\mathrm{gra}(L)$ is isomorphic to the result of uniformly substituting the square into the discrete two-vertex graph, as mentioned before. Note that in Fig. 3b the squares are drawn in a twisted way to suggest their isomorphism with $\mathrm{gra}(\{a, b\}) \leftarrow \mathrm{gra}(\{a, c\})$.

Suppose now that a graph g is represented by a grep language $L_0$ which is the concatenation $L_1L_2L_3$ of three other grep languages. Such concatenations are clearly related to "double" uniform substitutions, which have an extremely regular structure. Uniform substitution is associative, i.e., for graphs $g_1$, $g_2$, $g_3$, the graphs $(g_1 \leftarrow g_2) \leftarrow g_3$ and $g_1 \leftarrow (g_2 \leftarrow g_3)$ are isomorphic. In fact, both of them are isomorphic to the "double" uniform substitution, denoted $g_1 \leftarrow g_2 \leftarrow g_3$, that is defined to be the graph $(V, E)$ with

$$V = V_{g_1} \times V_{g_2} \times V_{g_3},$$

and for distinct $(x_1, y_1, z_1), (x_2, y_2, z_2) \in V$,

$$E((x_1, y_1, z_1), (x_2, y_2, z_2)) = \begin{cases} E_{g_1}(x_1, x_2) & \text{if } x_1 \neq x_2 \\ E_{g_2}(y_1, y_2) & \text{if } x_1 = x_2;\ y_1 \neq y_2 \\ E_{g_3}(z_1, z_2) & \text{if } x_1 = x_2;\ y_1 = y_2. \end{cases}$$

The isomorphisms between $(g_1 \leftarrow g_2) \leftarrow g_3$, $g_1 \leftarrow (g_2 \leftarrow g_3)$, and $g_1 \leftarrow g_2 \leftarrow g_3$ are the canonical ones between cartesian products.

As an example of a double uniform substitution, consider the graph of Fig. 3b. It is easily seen that this graph is isomorphic to the graph $g_1 \leftarrow g_2 \leftarrow g_1$, where $g_1$ is the discrete two-vertex graph and $g_2$ is the two-vertex one-edge graph.

It follows directly from Theorem 12 that $\mathrm{gra}(L_1L_2L_3)$ is isomorphic to $\mathrm{gra}(L_1) \leftarrow \mathrm{gra}(L_2) \leftarrow \mathrm{gra}(L_3)$, with concatenation as isomorphism. Thus, for the language $L = \{a, c\} \cdot \{a, b\} \cdot \{a, c\}$ of Example 2(2) and Fig. 3, $\mathrm{gra}(L)$ is isomorphic to the graph $g_1 \leftarrow g_2 \leftarrow g_1$ just mentioned, as also explained as an example just after Theorem 12.

These considerations can be generalized to the fact that $\mathrm{gra}(L_1 \ldots L_n)$ **isom** $\mathrm{gra}(L_1) \leftarrow \cdots \leftarrow \mathrm{gra}(L_n)$, in an obvious way. As an example, the graph $\mathrm{gra}(1 \cdot \{0, 1\}^n)$ of Example 2(1) is isomorphic to $h \leftarrow h \leftarrow \cdots \leftarrow h$ ($n$ times).

Let us finally consider the case that the language $L_0$ contains a concatenation of two languages $L_1$ and $L_2$ rather than being equal to it. Thus, let $L_1L_2 \subseteq L_0$, where $L_0$, $L_1$, $L_2$ represent the graphs $g_0$, $g_1$, $g_2$, respectively. Then Theorem 12 implies that $g_1 \leftarrow g_2$ can be embedded in $g_0$, i.e., that $g_0$ has a subgraph that is isomorphic to $g_1 \leftarrow g_2$. One might think that this also holds the other way around, i.e., that every graph $g_0$ that has a subgraph isomorphic with a uniform substitution $g_1 \leftarrow g_2$, can be represented by a grep language $L_0$ such that $L_1L_2 \subseteq L_0$, where $L_1$ and $L_2$ represent $g_1$ and $g_2$. This is, however, not true, as shown by the following counterexample.

EXAMPLE 13. Let $g_0$ be the graph in Fig. 8a, and let $g_1$ and $g_2$ both be the graph with two vertices and one edge. Obviously, $g_0$ has a subgraph isomorphic to $g_1 \leftarrow g_2$, viz. the subgraph induced by the four vertices in the square of Fig. 8a. We claim that there do not exist grep languages $L_0, L_1, L_2$ that represent $g_0, g_1, g_2$ such that $L_1L_2 \subseteq L_0$. Assume to the contrary that such languages do exist. Let $L_1 = \{u, v\}$ with $E_{\mathrm{gra}(L_1)} = \{(u, v)\}$, and $L_2 = \{x, y\}$ with $E_{\mathrm{gra}(L_2)} = \{(x, y)\}$, where $u, v, x, y$ are strings. Then $L_0 = \{ux, uy, vx, vy, z\}$ for some string $z$. By the definition of uniform substitution, $\mathrm{gra}(L_1) \leftarrow \mathrm{gra}(L_2)$ is the graph shown in Fig. 8b. Hence, by Theorem 12(1), $\mathrm{gra}(L_1L_2)$ is the graph shown in Fig. 8c. Since $\mathrm{gra}(L_1L_2)$ is isomorphic to a subgraph of $g_0$ and since it is easily checked that the graph of Fig. 8c can only be embedded in the graph of Fig. 8a in one way, $\mathrm{gra}(L_0)$ must be the graph shown in

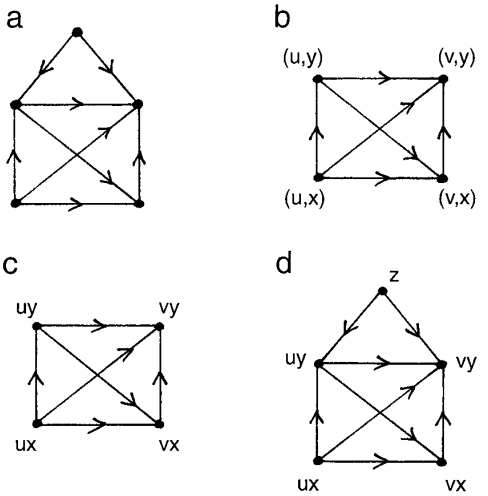


**FIG. 8.** Example 13.

Fig. 8d. Now, since $E_{\mathrm{gra}(L_0)}(z, ux) \neq E_{\mathrm{gra}(L_0)}(z, uy)$, $z$ has prefix $u$ by property (F3). However, since $E_{\mathrm{gra}(L_0)}(z, vx) \neq E_{\mathrm{gra}(L_0)}(z, vy)$, $z$ has prefix $v$. This contradicts the fact that $L_1$ is prefix-free.

It turns out that the embedding of $g_1 \leftarrow g_2$ into $g_0$ has a very special property: it transforms the rows of $g_1 \leftarrow g_2$ (which are clans of $g_1 \leftarrow g_2$) into "clan-separable" subgraphs of $g_0$. Let $g$ be a graph and $X$, $Y \subseteq V_g$; we say that $X$ and $Y$ are *clan-separable* if there exist disjoint clans $X'$ and $Y'$ of $g$ such that $X \subseteq X'$ and $Y \subseteq Y'$. Note that in Example 13 (Fig. 8b) the "rows" $\{ux, uy\}$ and $\{vx, vy\}$ are not clans, and hence they are not clan-separable (because $z$ is the only vertex left).

THEOREM 14. *Let $L_0, L_1, L_2$ be grep languages over $h$ such that $L_1 L_2 \subseteq L_0$, and let $g_i = \mathrm{gra}(L_i)$ for $i = 1, 2, 3$. Then there is an isomorphism $\phi$ from $g_1 \leftarrow g_2$ to a subgraph of $g_0$ such that $\phi((g_1 \leftarrow g_2)(x, *))$ and $\phi((g_1 \leftarrow g_2)(y, *))$ are clan-separable for all distinct $x$, $y \in V_{g_1}$.*

*Proof.* Since $L_1 L_2 \subseteq L_0$, the concatenation function $\phi$ is an isomorphism from $g_1 \leftarrow g_2$ to the subgraph $\mathrm{gra}(L_1 L_2)$ of $g_0$ by Theorem 12. For every $x \in L_1$, $\phi((g_1 \leftarrow g_2)(x, *)) = xL_2 \subseteq \mathrm{pref}(x, L_0)$. Since $L_1$ is prefix-free, $\mathrm{pref}(x, L_0)$ and $\mathrm{pref}(y, L_0)$ are disjoint clans of $L_0$ for all distinct elements $x$, $y$ of $L_1$ (cf. Lemma 3). Hence $\phi((g_1 \leftarrow g_2)(x, *))$ and $\phi((g_1 \leftarrow g_2)(y, *))$ are clan-separable. ∎

It can be shown that the clan separability of the rows of $g_1 \leftarrow g_2$ in $g_0$ is in fact also a sufficient condition, i.e., it guarantees the existence of an alphabet graph $h$ and grep languages $L_i$ over $h$ that represent $g_i$, such that $L_1 L_2 \subseteq L_0$.

## CONCLUSION

We have introduced a new method for the specification of (finite) graphs: by finite prefix-free languages over an alphabet with a graph structure. Our results have demonstrated the intrinsic relationship between language concatenation and graph substitution. We have also demonstrated that the method naturally generalizes the hierarchical specification of various well-known classes of graphs.

We see this paper as the beginning of a systematic study of this new graph specification method. Let us mention some topics that in our opinion should be investigated.

In general, as observed in the paper, we wish to know how the structure of the grep language influences the structure of the represented graph, and, in particular, how well-known operations on languages are reflected as operations on graphs. For example, it would be interesting to investigate the (language) operations of homomorphism and substitution, both closely related to concatenation. Another example is reversal: if a grep language is both prefix-free and suffix-free, then what is the relationship between the two graphs that are represented by the language and its reversal?

Another, related question is how operations on the alphabet graph influence the corresponding classes of represented graphs. As a concrete example, if there is a graph homomorphism from $h_1$ to $h_2$, then what is the relationship between $\mathrm{Rep}(h_1)$ and $\mathrm{Rep}(h_2)$?

It is also of interest to know how properties of $h$ lead to properties of $\mathrm{Rep}(h)$; cf. Theorem 9. As an example, if $h$ is a comparability graph then so are all graphs in $\mathrm{Rep}(h)$. This follows from Theorem 8 and the fact that the class of comparability graphs is closed under graph substitution (see Theorem 5.6 of [Gol]).

Is there an efficient algorithm that computes for a given graph $g$ the smallest representation $\mathrm{gra}_h(L)$ of $g$? Here "smallest" refers to the sum of the sizes of $h$ and the minimal automaton that recognizes $L$. It would also be of interest to investigate the properties of the size of the smallest representation of $g$, viewed as a complexity measure.

We have restricted ourselves to finite languages, but clearly the definitions also apply to infinite languages. Thus, a natural method for the specification of infinite graphs is obtained. This allows one to use language theory in its generality for the investigation of infinite graphs. Thus, one may ask which infinite graphs are regular in the sense that they can be specified by regular grep languages, and the same question can be asked for context-free grep languages. The resulting notions of regular and context-free infinite graphs should be compared to the existing ones in the literature [MulSch, Cou2].

It would be interesting to try to generalize our method to hypergraphs. To decide when a finite number of strings (representing vertices of a hypergraph) should form a hyperedge, one would have to generalize the notion of "first difference" from two strings to any number of strings and change the notion of "alphabet graph" accordingly. Again, it would be natural to do this generalization in such a way that language concatenation would still correspond to hypergraph substitution (as used, e.g., in the area of graph grammars).

## ACKNOWLEDGMENT

We thank the referees for fruitful suggestions.

## REFERENCES

[AdhPen] G. S. Adhar and S. Peng, Parallel algorithms for cographs; recognition and applications, *in* "Proceedings, WADS '89" (F. Dehne, J.-R. Sack, and N. Santoro, Eds.), Lecture Notes in Computer Science Vol. 382, pp. 335–351, Springer-Verlag, Berlin, 1989.

[Arn] S. Arnborg, Efficient algorithms for combinatorial problems on graphs with bounded decomposability—A survey, BIT **25** (1985), 2–23.

[ArnLagSee] S. Arnborg, J. Lagergren, and D. Seese, Easy problems for tree-decomposable graphs, *J. Algorithms* **12** (1991), 308–340.

[BodMöh] H. L. Bodlaender and R. H. Möhring, The pathwidth and treewidth of cographs, *in* "Proceedings, SWAT 90" (J. R. Gilbert and R. Karlsson, Eds.), Lecture Notes in Computer Science, Vol. 447, pp. 301–309, Springer-Verlag, Berlin, 1990.

[Brz] J. A. Brzozowski, Derivatives of regular expressions, *J. Assoc. Comput. Mach.* **11** (1964), 481–494.

[BueMöh] H. Buer and R. H. Möhring, A fast algorithm for the decomposition of graphs and posets, *Math. Oper. Res.* **8** (1983), 170–184.

[CorLerSte] D. G. Corneil, H. Lerchs, and L. Stewart Burlingham, Complement reducible graphs, *Discrete Appl. Math.* **3** (1981), 163–174.

[Cou1] B. Courcelle, An axiomatic definition of context-free rewriting and its application to NLC graph grammars, *Theoret. Comput. Sci.* **55** (1987), 141–181.

[Cou2] B. Courcelle, The monadic second-order logic of graphs. II. Infinite graphs of bounded width, *Math. Systems Theory* **21** (1989), 187–221.

[Cou3] B. Courcelle, The monadic second-order logic of graphs. III. Tree-width, forbidden minors, and complexity issues, *RAIRO Inform. Théor. Appl.* **26** (1992), 257–286.

[EhrRoz1] A. Ehrenfeucht and G. Rozenberg, Theory of 2-structures. Part I. Clans, basic subclasses, and morphisms, *Theoret. Comput. Sci.* **70** (1990), 277–303.

[EhrRoz2] A. Ehrenfeucht and G. Rozenberg, Theory of 2-structures. Part II. Representation through labeled tree families, *Theoret. Comput. Sci.* **70** (1990), 305–342.

[EhrRoz3] A. Ehrenfeucht and G. Rozenberg, Primitivity is hereditary for 2-structures, *Theoret. Comput. Sci.* **70** (1990), 343–358.

[EhrKreRoz] H. Ehrig, H.-J. Kreowski, and G. Rozenberg (Eds.), "Graph Grammars and their Application to Computer Science," Lecture Notes in Computer Science, Vol. 532, Springer-Verlag, Berlin, 1991.

[Eil] S. Eilenberg, "Automata, Languages, and Machines," Vol. A, Academic Press, New York, 1974.

[Eng] J. Engelfriet, Context-free NCE graph grammars, *in* "Proceedings, FCT'89" (J. Csirik, J. Demetrovics, and F. Gécseg, Eds.), Lecture Notes in Computer Science, Vol. 380, pp. 148–161, Springer-Verlag, Berlin, 1989.

[EngHarProRoz] J. Engelfriet, T. Harju, A. Proskurowski, and G. Rozenberg, "Characterization and Complexity of Uniformly Non-primitive Labeled 2-Structures," Report 94-31, Leiden University, 1994; *Theoret. Comput. Sci.*, to appear.

[EngRoz] J. Engelfriet and G. Rozenberg, Graph grammars based on node rewriting: An introduction to NLC graph grammars, *in* [EhrKreRoz pp. 12–23].

[Gol] M. C. Golumbic, "Algorithmic Graph Theory and Perfect Graphs," Academic Press, San Diego, 1980.

[Har] F. Harary, "Graph theory," Addison–Wesley, Reading, MA, 1969.

[HopUll] J. E. Hopcroft and J. D. Ullman, "Introduction to Automata Theory, Languages, and Computation," Addison–Wesley, Reading, MA, 1979.

[JanRoz] D. Janssens and G. Rozenberg, Neighbourhood-uniform NLC grammars, *Comput. Vision Graphics Image Process.* **35** (1986), 131–151.

[MöhRad] R. H. Möhring and F. J. Radermacher, Substitution decomposition for discrete structures and connections with combinatorial optimization, *Ann. Discrete Math.* **19** (1984), 257–356.

[MulSch] D. E. Muller and P. E. Schupp, The theory of ends, pushdown automata, and second-order logic, *Theoret. Comput. Sci.* **37** (1985), 51–75.

[MulSpi] J. H. Muller and J. Spinrad, Incremental modular decomposition, *J. Assoc. Comput. Mach.* **36** (1989), 1–19.

[RabSco] M. O. Rabin and D. Scott, Finite automata and their decision problems, IBM *J. Res. Develop.* **3** (1959), 114–125.

[RobSey] N. Robertson and P. Seymour, Graph Minors. II. Algorithmic aspects of tree-width, *J. Algorithms* **7** (1986), 309–322.

[Sab1] G. Sabidussi, The composition of graphs, *Duke Math. J.* **26** (1959), 693–696.

[Sab2] G. Sabidussi, The lexicographic product of graphs, *Duke Math. J.* **28** (1961), 573–578.

[Sab3] G. Sabidussi, Graph derivatives, *Math. Z.* **76** (1961), 385–401.

[ValTarLaw] J. Valdez, R. E. Tarjan, and E. Lawler, The recognition of series parallel digraphs, *SIAM J. Comput.* **11** (1982), 298–313.